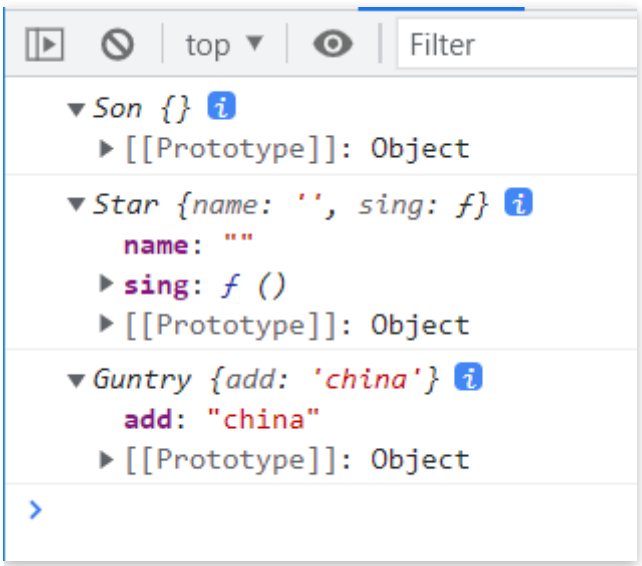
1 继承

```
<script>
        //  1.通过原型继承
        function Person(name, age) {
            this.name = name;
            this.age = age
            this.say = function () {
                console.log('我是演员');
            }
        }
        let p = new Person();

        function Son() {

        }
        Son.prototype = Person.prototype;//这样直接赋值会有问题,如果修改了子原型对象,父原型对象也会跟着一起变化
        let ldh = new Son();
        console.log(ldh);

        // 2 方法劫持
        function Star() {
            this.name = name;
            this.sing = function () {
                console.log(this.sing);
            }
        }

        function Ming() {
            Star.call(this, arguments);
        }

        let zjl = new Star();
        console.log(zjl);

        // 类的继承
        class Guntry {
```

```
            constructor() {
                this.name = name;
            }
        }

        class Area extends Guntry {
            constructor(name, area) {
                super(name);// super调用父级的方法，要在子类方法前调用
                this.area = area;
            }
        }

        // 4 直接指定原型
        let obj = Object.create(Guntry.prototype);
        obj.add = 'china';
        console.log(obj);
    </script>
```



2 继承

```
<script>
        // 借用父构造函数继承属性
        // 1. 父构造函数
        function Father(uname, age) {
            // this 指向父构造函数的对象实例
            this.uname = uname;
```

```
            this.age = age;
        }
        Father.prototype.money = function () {
            console.log(100000);

        };
        // 2 .子构造函数
        function Son(uname, age, score) {
            // this 指向子构造函数的对象实例
            Father.call(this, uname, age);
            this.score = score;
        }
        // Son.prototype = Father.prototype;  这样直接赋值会有问题,如果修改了子原型对象,父原型对象也会跟着一起变化
        Son.prototype = new Father();
        // 如果利用对象的形式修改了原型对象,别忘了利用constructor 指回原来的构造函数
        Son.prototype.constructor = Son;
        // 这个是子构造函数专门的方法
        Son.prototype.exam = function () {
            console.log('孩子要考试');

        }
        var son = new Son('刘德华', 18, 100);
        console.log(son);
        console.log(Father.prototype);
        console.log(Son.prototype.constructor);
    </script>
```

JS： Proxy

1.概述

Proxy 用于修改某些操作的默认行为，等同于在语言层面做出修改，所以属于一种“元编程”（meta programming），即对编程语言进行编程。

Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。Proxy 这个词的原意是代理，用在这里表示由它来“代理”某些操作，可以译为“代理器”。

2.语法

```
let p = new Proxy(target, handler);
```

3.参数

target：需要使用Proxy包装的目标对象（可以是任何类型的对象，包括原生数组，函数，甚至另一个代理）。

handler: 一个对象，其属性是当执行一个操作时定义代理的行为的函数(可以理解为某种触发器)。具体的handler相关函数请查阅官网

下面是使用示例，一个简单的代理:

```
let obj = {
            a: 1
        };

let proxyObj = new Proxy(obj, {
    get(target, proKey) {
         return target[proKey];
    },
 });
 console.log(proxyObj.a);

let validator = {
     set(target, key, value) {
     if (key === 'age') {
          if (!Number.isInteger(value) || value > 150) {
               throw new Error('数据不合法')
           }
       }
    target[key] = value;
    }
}

        let proxy1 = new Proxy({}, validator);
        proxy1.age = 89;
        console.log(proxy1.age);

        proxy1.age = 1000;
        console.log(1 + +"2" + "2");
```